Wykonał: Jakub Nowakowski

1. Cele doświadczenia

Doświadczenie zostało wykonane w celu porównania czasów działania algorytmów Jarvisa i Grahama, jako algorytmów wyznaczających otoczkę wypukłą zadanego zbioru punktów.

2. Konfiguracja stanowiska

Procesor: intel Corei7-8750H
Wersja Python'a: Python 3.7.4

3. Wstęp teoretyczny i przewidywania dotyczące wyników doświadczenia

Zarówno algorytm Jarvisa jak i algorytm Grahama służą do wyznaczania otoczki wypukłej zbioru punktów, jednak sposób w jaki to robią jest różny dla obu algorytmów co powoduje różnice w czasie ich działania.

Algorytm Grahama w celu wyznaczenia otoczki:

- wybiera punkt początkowy O(n)
- sortuje punkty względem współrzędnej biegunowej O(nlogn)
- redukuje liczbę punktów O(n)
- inicjalizuje stos O(1)
- przegląda wszystkie punkty O(n-3)

Ostatecznie otrzymujemy złożoność czasową O(nlon), widzimy więc że nie jest ona zależna od liczby punktów należących do otoczki.

Algorytm Jarvisa natomiast wykonuje następujące kroki:

- Wybiera punkt początkowy O(n)
- Wybiera kolejny punkt otoczki O(k*n) (liniowe poszukiwanie kolejnego punktu O(n) powtarzane k razy, gdzie k to liczba punktów należących do otoczki)

Otrzymujemy więc złożoność czasową O(k*n), jest ona uzależniona od liczby punktów należących do otoczki i dla k<<n algorytm ten ma złożoność liniową.

Można się więc spodziewać, że algorytm Jarvisa będzie działał szybciej, gdy k<nlogn, a więc dla zbiorów, dla których liczba punktów należących do otoczki jest znacznie mniejsza od ogólnej liczby punktów.

4. **Metodyka**

Oba algorytmy były testowane na losowo wygenerowanych zbiorach, a następnie porównywane były czasy działania obu algorytmów (porównanie odbywa się pomiędzy algorytmami dla kopi tego samego zbioru punktów).

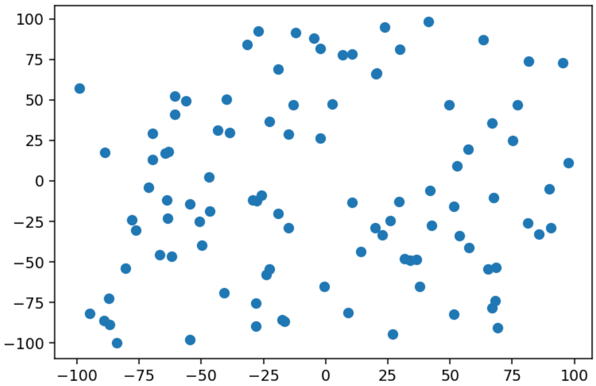
5. **Zbiory, na których został przeprowadzony eksperyment**

Zbory były generowane za pomocą funkcji umożliwiających zmiany parametrów generowania. Testy zostały przeprowadzone na zbiorach które możemy podzielić na następujące klasy:

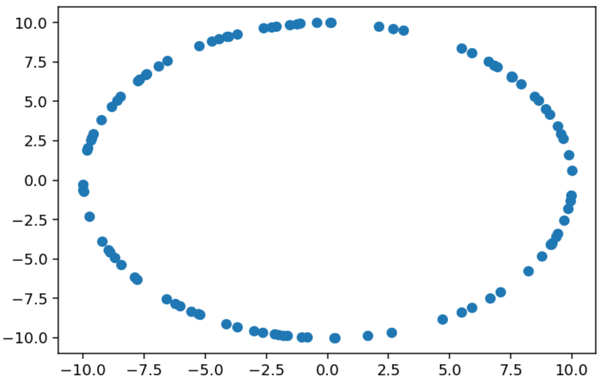
A. Test dla zbioru n losowych punktów o współrzędnych z przedziału [a,b]
B. Test dla n punktów leżących na okręgu o promieniu R o środku w punkcie (x,y)
C. Test dla n punktów leżących na prostokącie
D. Test dla n1 punktów leżących na bokach kwadratu i n2 leżących na jego przekątnych (zbiór zawiera wierzchołki kwadratu)

Współrzędne punktów zostały wygenerowane za pomocą funkcji uniform() z biblioteki random.

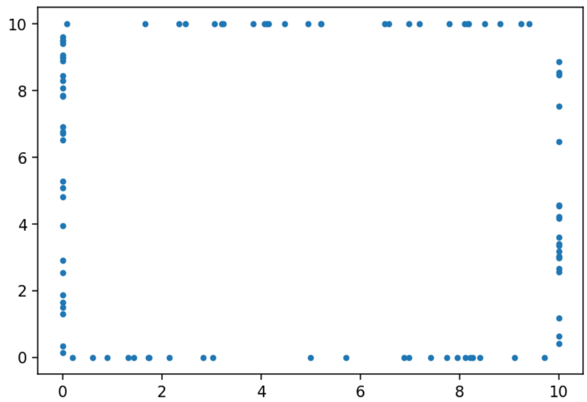
Wizualizacje przykładowych zbiorów:



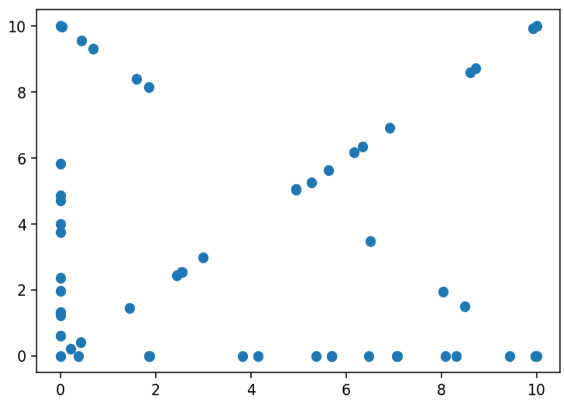
Rysunek 1 Przykładowy zbiór klasy A



Rysunek 2 Przykładowy zbiór klasy B



Rysunek 3 Przykładowy zbiór klasy C



Rysunek 4 Przykładowy zbiór klasy D

6. Wyniki przeprowadzonych testów

• Zbiory klasy A

Możliwe problemy ze zbiorem: brak konkretnych problemów

Zbiory tej klasy zostały wygenerowane w sposób losowy, a jedynym ograniczeniem przy generowaniu punktów był zakres współrzędnych oznaczony jako[a,b] oraz liczba punktów należących do zbiorów. Ta klasa zbiorów nie powinna stwarzać problemów przy testowaniu.

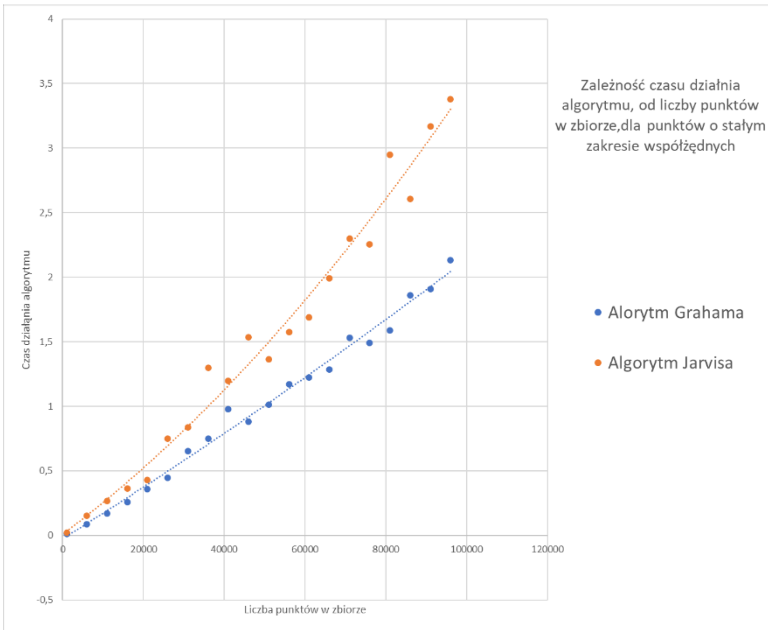
Epsilon obrane dla testów e=1e-12

Zależność czasu działania algorytmów w zależności od liczby punktów w zbiorze dla [a,b] = [-1000,1000],oraz zależność czasu działania obu algorytmów przy [a,b] = [-n,n]

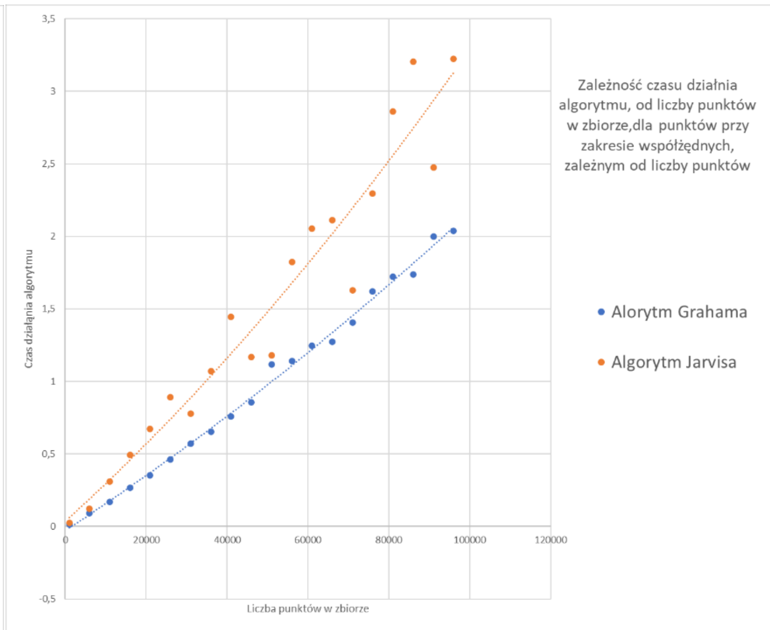
Czasy dla [A,B] = [-1000,1000]			
n	**Liczba punktów na otoczce**	**Alorytm Grahama czas działania [s]**	**Algorytm Jarvisa czas działania [s]**
1000	17	0,013	0,020
6000	25	0,087	0,151
11000	25	0,170	0,269
16000	23	0,259	0,366
21000	21	0,360	0,429
26000	27	0,448	0,750
31000	28	0,655	0,838
36000	31	0,748	1,299
41000	27	0,977	1,198
46000	29	0,882	1,534
51000	26	1,013	1,363
56000	29	1,171	1,575
61000	27	1,224	1,691
66000	27	1,285	1,992
71000	28	1,532	2,300
76000	30	1,490	2,257
81000	32	1,587	2,947
86000	29	1,860	2,607
91000	33	1,910	3,167
96000	33	2,131	3,378

Czasy dla [A,B] = [-n,n]			
n	**Liczba punktów na otoczce**	**Alorytm Grahama czas działania [s]**	**Algorytm Jarvisa czas działania [s]**
1000	22	0,013	0,026
6000	21	0,090	0,124
11000	29	0,168	0,308
16000	31	0,266	0,494
21000	31	0,352	0,673
26000	31	0,464	0,891
31000	24	0,571	0,778
36000	29	0,651	1,070
41000	35	0,757	1,446
46000	26	0,855	1,167
51000	23	1,117	1,180
56000	33	1,139	1,824
61000	33	1,244	2,055
66000	30	1,275	2,114
71000	23	1,404	1,627
76000	30	1,620	2,297
81000	34	1,721	2,863
86000	34	1,738	3,207
91000	27	1,999	2,475
96000	33	2,039	3,225

Tabela 1 Czasy działania algorytmów dla zbiorów klasy A

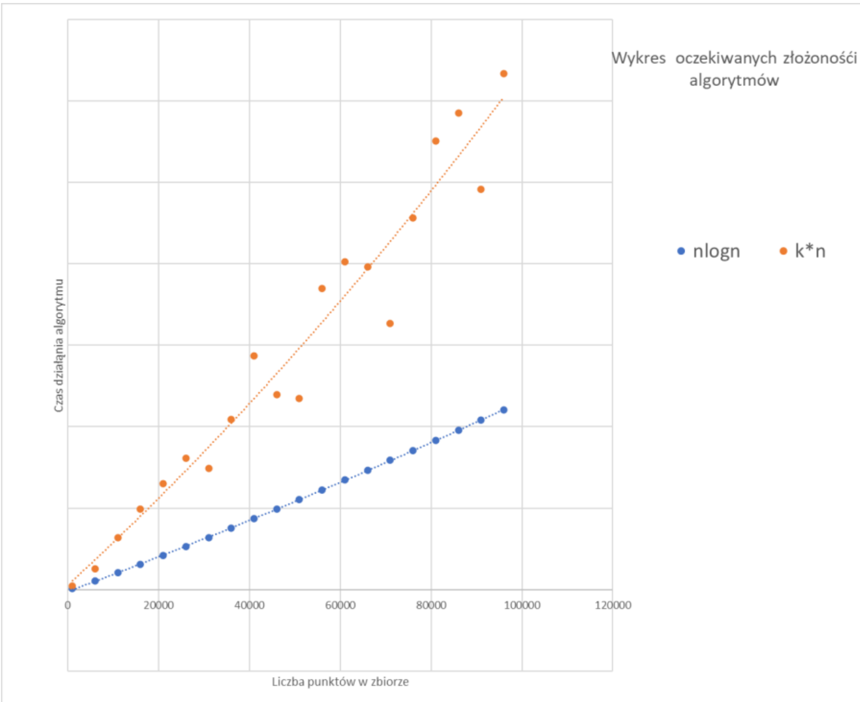


Wykres 1



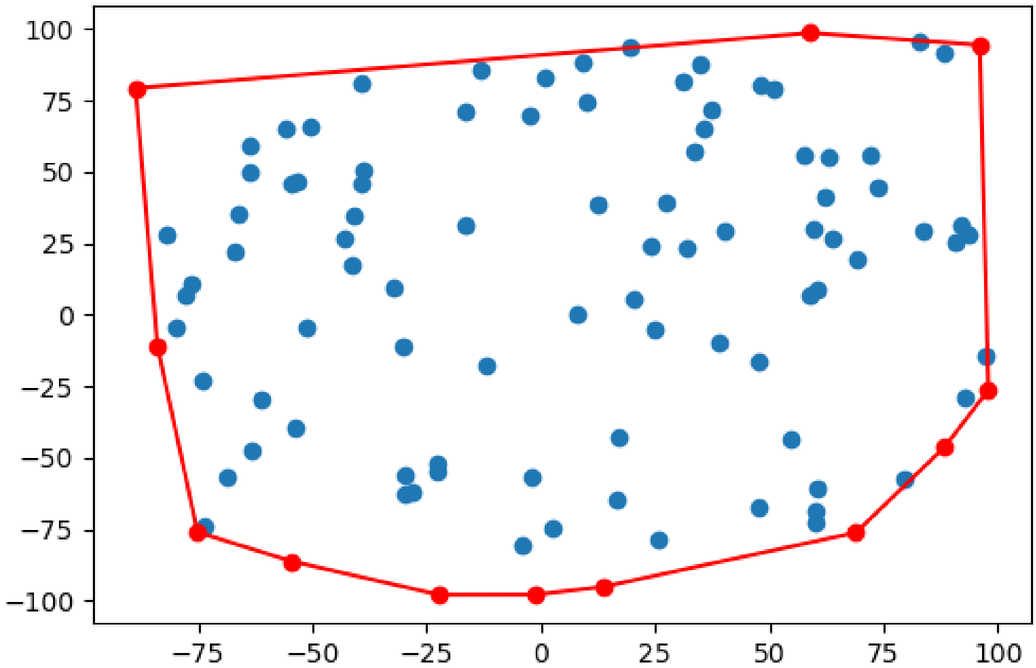
Wykres 2

Możemy zauważyć, że wpływ rozproszenia punktów na działania algorytmów jest prawie nie zauważalny, ponadto dla losowego zbioru liczba punktów tworzących otoczkę również zależy głównie od liczby punktów w zbiorze. Widać również, że dla losowych punktów szybciej działa algorytm Grahama, oznacza to, że liczba punktów na otoczce rośnie szybciej niż logn.



Wykres 3

Widać również, że czasy działania algorytmów wyglądają podobnie jak wynikałoby to ze złożoności czasowej, dlatego też można przypuszczać, że oba zostały zaimplementowane poprawnie.



Rysunek 5 Przykładowa otoczka dla zbioru klasy A

- **Zbiory klasy B**
  Możliwe problemy ze zbiorem: program może nie zaliczyć całego zbioru do otoczki

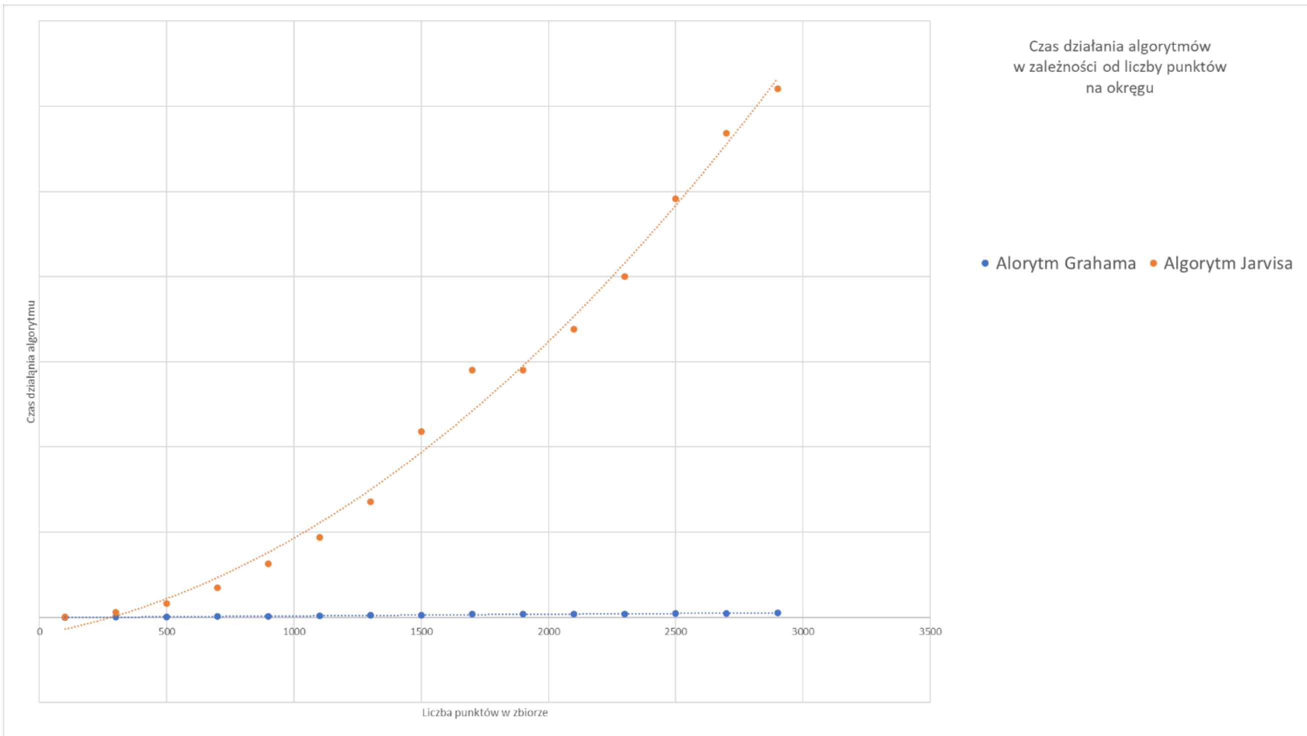
Punkty w zbiorach tej klasy zostały wygenerowane na okręgach o promieniu R w środku w punkcie (x_0,y_0). Wiemy, że dla takiego zbioru wszystkie punkty należą do jego otoczki wypukłej, co jest przypadkiem pesymistycznym dla algorytmu Jarvisa, osiąga on wtedy złożoność O(n^2) ponieważ k = n. Algorytm Grahama powinien działać szybciej dla zbiorów tej klasy.

Epsilon obrane dla testów e=1e-12

Czas działania algorytmów był badany dla zbiorów wygenerowanych przy następujących parametrach:
- Środek okręgu w punkcie (0,0)
- n liczba punktów w zbiorze
- n/2 promień okręgu

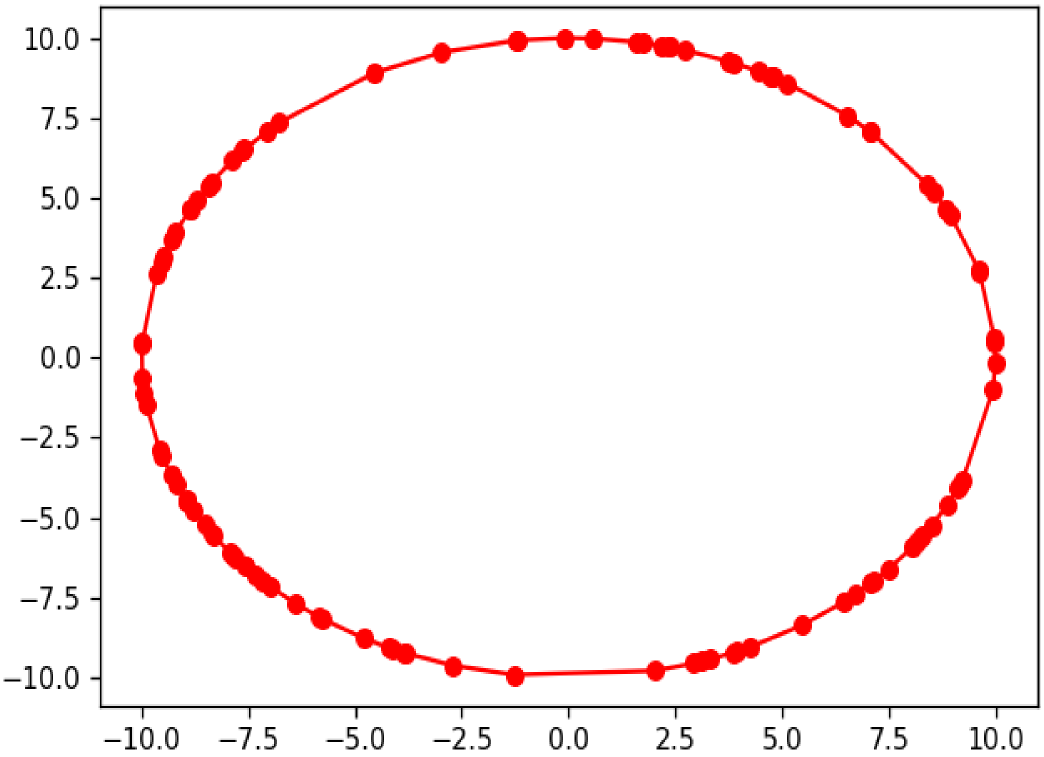
n	Liczba punktów na otoczce	Algorytm Grahama czas działania [s]	Algorytm Jarvisa czas działania [s]
100	100	0,001964	0,015993
300	300	0,007972	0,117733
500	500	0,012967	0,32109
700	700	0,018947	0,693568
900	900	0,029935	1,25943
1100	1100	0,031904	1,877364
1300	1300	0,048867	2,715309
1500	1500	0,051821	4,36246
1700	1700	0,077792	5,810225
1900	1900	0,077307	5,808164
2100	2100	0,080785	6,763904
2300	2300	0,07579	7,999223
2500	2500	0,094747	9,823156
2700	2700	0,096583	11,37317
2900	2900	0,109675	12,41061

Tabela 2 Czasy otrzymane w testach zbiorów klasy B



Wykres 4

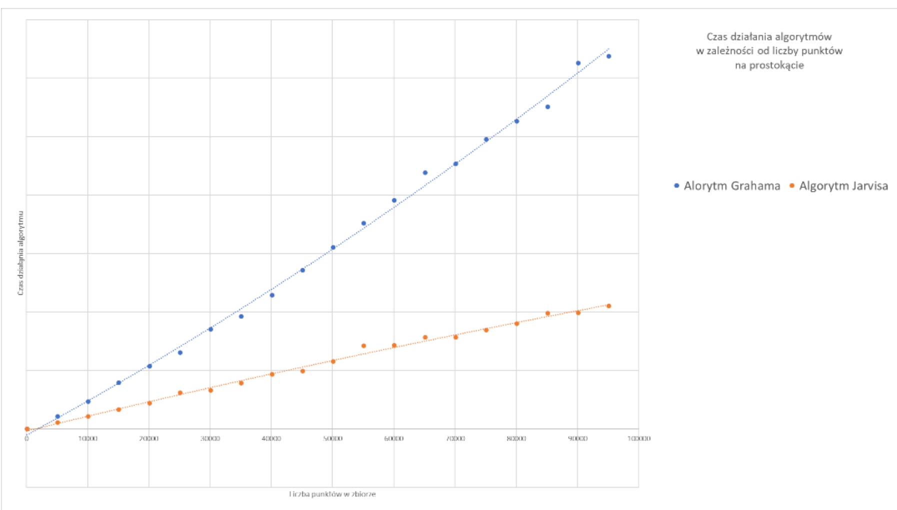
Zgodnie z przewidywaniami algorytm Jarvisa jest znacznie wolniejszy od Algorytmu Grahama dla zbioru zawierającego punkty leżące na okręgu. Widać też, że jego złożoność czasowa jest kwadratowa kiedy złożoność algorytmu Grahama jest znacznie niższa.



Rysunek 6 Przykładowa otoczka dla zbioru klasy B

- **Zbiory klasy C**
  Możliwe problemy ze zbiorem: program może zaliczyć do otoczki punkty współliniowe
  Zbiory tej klasy zostały wygenerowane na bokach prostokąta, dlatego przy założeniu, że na każdym boku są przynajmniej 2 punkty możemy stwierdzić, że otoczka tych zbiorów powinna liczyć zawsze 8 punktów. Dla tych zbiorów lepszy powinien okazać się algorytm Jarvisa ponieważ osiąga wtedy złożoność O(8n), a jego przewaga powinna wzrastać dla rosnącej liczby punktów.

  Epsilon obrane dla testów e=1e-12

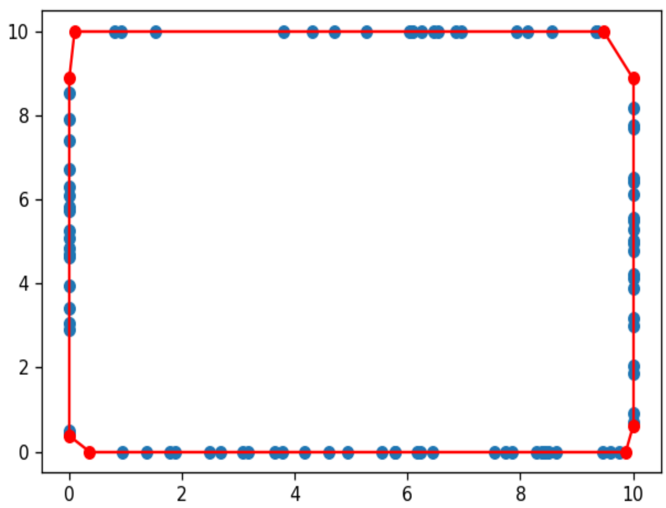


Wykres 5

n	Liczba punktów na otoczce	Algorytm Grahama	Algorytm Jarvisa
100	8	0,004	0,000
5100	8	0,215	0,109
10100	8	0,471	0,215
15100	8	0,794	0,331
20100	8	1,075	0,441
25100	8	1,305	0,619
30100	8	1,706	0,662
35100	8	1,924	0,786
40100	8	2,290	0,935
45100	8	2,713	0,990
50100	8	3,104	1,152
55100	8	3,517	1,422
60100	8	3,909	1,430
65100	8	4,384	1,565
70100	8	4,536	1,569
75100	8	4,948	1,693
80100	8	5,259	1,799
85100	8	5,507	1,980
90100	8	6,256	1,988
95100	8	6,376	2,104

Tabela 3 Czasy otrzymane w testach dla zbiorów klasy C

Zgodnie z przewidywaniami dla tej klasy zbiorów algorytm Jarvisa okazał się znacznie wydajniejszym rozwiązaniem. Dlatego też dla zbiorów, w których k << n algorytm Jarvisa jest rozsądniejszym rozwiązaniem niż algorytm Grahama.



Rysunek 6 Przykładowa otoczka dla zbioru klasy C

- ## Zbiory klasy D

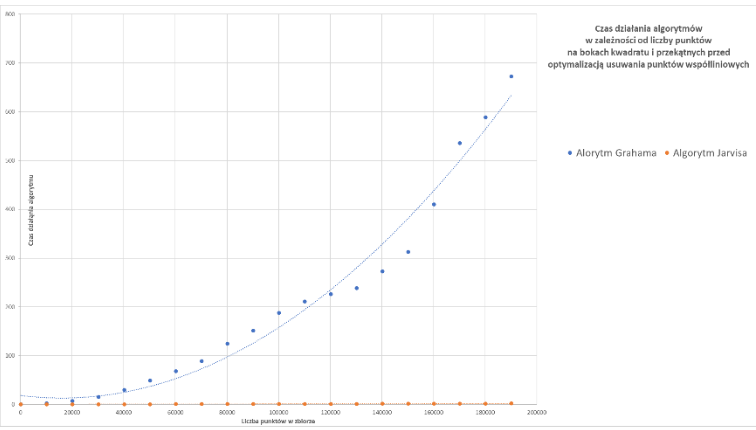
Możliwe problemy ze zbiorem: program może zaliczyć do otoczki punkty inne niż wierzchołki kwadratu

Punkty w tej klasie zbiorów zostały wygenerowane na bokach kwadratu, oraz jego przekątnych ponad to mamy gwarancję, że w zbiorze znajdują się wierzchołki tego kwadratu. Dla tego zbioru prawdopodobnie szybszy okażę się algorytm Jarvisa, ale tylko w przypadkach, kiedy liczba punktów zredukowanych przy usuwaniu tych o tej samej współrzędnej liniowej będzie znacznie większa od 4. Warto zauważyć, że punkty które zostają po precesje usuwania punktów współliniowych to tylko wierzchołki i jedna przekątna kwadratu.

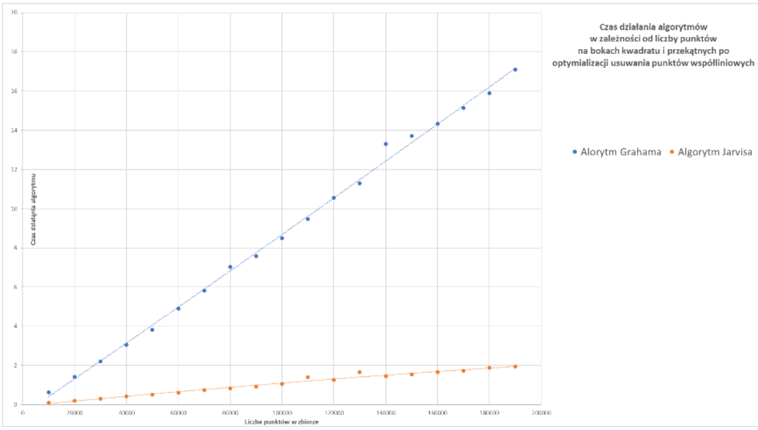
Epsilon obrane dla testów e=1e-12

Czas działania algorytmów był badany dla zbiorów wygenerowanych przy następujących parametrach:
- n1 punkty na bokach kwadratu
- n2 punkty na przekątnych kwadratu

Testy wykazały bardzo słabą wydajność algorytmu Grahama dla tego zbioru, winna może być metoda remove() usuwająca punkty z listy ponieważ działa ona w sposób liniowy i usuwając dużą część punktów z tego zbioru sprawia, że algorytm Grahama staje się kwadratowy, dla porównania zamieszczam wynik testów również dla algorytmu Grahama, który zamiast funkcji remove() przepisuje punkty nie współliniowe do nowej listy aktualizując w ten sposób zbiór punktów. Można również wykorzystać słowo kluczowe del to rozwiązanie jest nieznacznie wolniejsze od przepisania listy, ale również dużo szybsze od remove (). Po przeanalizowaniu, wyników testów dla poprzednich klas zbiorów po aktualizacji kodu można dojść do wniosku, że ta zmiana w kodzie nie miała znaczącego wpływu na czasy działania algorytmów na tych zbiorach.



Wykres 6 Czas działania dla zbiorów klasy C przed optymalizacją usuwania



Wykres 7 Czas działania dla zbiorów klasy C po optymalizacji usuwania

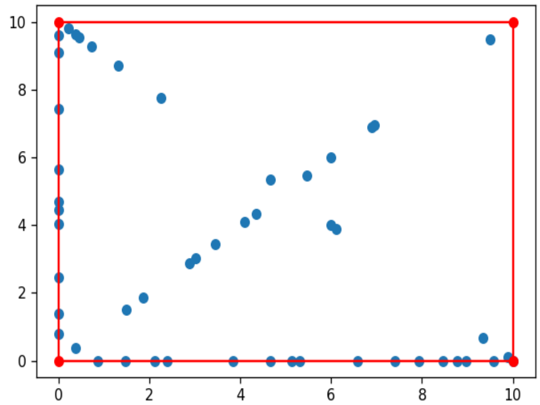
Czasy działania dla zbiorów klasy C przed optymalizacją usuwania				
n1	n2	Punkty na otoczce	Alorytm Grahama	Algorytm Jarvisa
5100	5100	4	2,113	0,115
10100	10100	4	7,344	0,244
15100	15100	4	15,463	0,389
20100	20100	4	29,615	0,539
25100	25100	4	49,074	0,618
30100	30100	4	68,022	0,763
35100	35100	4	89,040	0,798
40100	40100	4	124,786	0,986
45100	45100	4	151,117	1,073
50100	50100	4	187,724	1,186
55100	55100	4	211,095	1,195
60100	60100	4	226,375	1,270
65100	65100	4	238,898	1,351
70100	70100	4	273,098	1,451
75100	75100	4	312,729	1,553
80100	80100	4	410,413	1,887
85100	85100	4	535,988	1,802
90100	90100	4	588,680	1,962
95100	95100	4	672,317	2,091

Czasy działania dla zbiorów klasy C po potymalizacji usuwania				
n1	n2	Punkty na otoczce	Alorytm Grahama	Algorytm Jarvisa
5000	5000	4	0,640	0,101
10000	10000	4	1,420	0,211
15000	15000	4	2,215	0,313
20000	20000	4	3,054	0,421
25000	25000	4	3,821	0,516
30000	30000	4	4,902	0,623
35000	35000	4	5,809	0,758
40000	40000	4	7,042	0,833
45000	45000	4	7,583	0,932
50000	50000	4	8,502	1,068
55000	55000	4	9,467	1,400
60000	60000	4	10,561	1,264
65000	65000	4	11,298	1,666
70000	70000	4	13,300	1,455
75000	75000	4	13,704	1,542
80000	80000	4	14,321	1,659
85000	85000	4	15,132	1,738
90000	90000	4	15,896	1,893
95000	95000	4	17,098	1,954

Tabela 2 Wyniki testów dla zbiorów klasy D

Jak widać na tym przykładzie nawet pozornie mało istotne elementy implementacji mogą mieć ogromny wpływ na czas działania programu.

Rysunek 7 Przykładowa otoczka dla zbioru klasy D



7. Wnioski z doświadczenia

- Algorytm Grahama lepiej sprawdza się dla zbiorów, w których duża część zbioru należy do otoczki
- Dla losowo wybranych punktów szybszy jest algorytm Grahama
- Gdy liczba punktów na otoczce jest mała opłaca się skorzystać z algorytmu Jarvisa który jest wtedy dużo szybszy od algorytmu Grahama
- O ile w przypadku losowych punktów różnice pomiędzy algorytmem Grahama a Jarvisa są niewielkie, dla pesymistycznego przypadku (np. okrąg) algorytm Jarvisa jest znacznie mniej wydajny
- Posiadając informacje o ułożeniu punktów możemy dobrać taki algorytm, aby znajdowanie otoczki było szybsze